# autogluon_each_location

October 18, 2023

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = None
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 0
num_bag_sets = 0

use_tune_data = True
use_test_data = True
tune_and_test_length = 24*30*6 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False
sample_weight_estimated = 1

run_analysis = False
```

[2]:
```
import pandas as pd
import numpy as np
```

```python
import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00, we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
       'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
       'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1 hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]



    date_calc = None
    if "date_calc" in X.columns:
```

```
        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
```

```python
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
 ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)
```

```python
    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
 ↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
```

```
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

# 1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
 ↪inplace=True)
```

```python
for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())
```

```python
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups), repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())
```

```python
to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms", "dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm", "wind_speed_u_10m:ms", "wind_speed_v_10m:ms"]

X_train.drop(columns=to_drop, inplace=True)
```

```python
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                     7.700              1.22825
2019-06-02 23:00:00                     7.700              1.22350
2019-06-03 00:00:00                     7.875              1.21975
2019-06-03 01:00:00                     8.425              1.21800
2019-06-03 02:00:00                     8.950              1.21800

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951               0.000000
2019-06-02 23:00:00           1689.824951               0.000000
2019-06-03 00:00:00           1563.224976               0.000000
2019-06-03 01:00:00           1283.425049            6546.899902
2019-06-03 02:00:00           1003.500000          102225.898438

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00             0.00       1728.949951              0.0
2019-06-02 23:00:00             0.00       1689.824951              0.0
2019-06-03 00:00:00             0.00       1563.224976              0.0
2019-06-03 01:00:00             0.75       1283.425049              0.0
2019-06-03 02:00:00            23.10       1003.500000              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 22:00:00      280.299988          0.000          0.000000  …
2019-06-02 23:00:00      280.299988          0.000          0.000000  …
2019-06-03 00:00:00      280.649994          0.000          0.000000  …
2019-06-03 01:00:00      281.674988          0.300       7743.299805  …
2019-06-03 02:00:00      282.500000         11.975      60137.601562  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 22:00:00   286.225006           100.000000  40386.476562
2019-06-02 23:00:00   286.899994           100.000000  33770.648438
2019-06-03 00:00:00   286.950012           100.000000  13595.500000
2019-06-03 01:00:00   286.750000           100.000000   2321.850098
2019-06-03 02:00:00   286.450012            99.224998  11634.799805

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 22:00:00              3.600               -3.575
```

```
2019-06-02 23:00:00              3.350              -3.350
2019-06-03 00:00:00              3.050              -2.950
2019-06-03 01:00:00              2.725              -2.600
2019-06-03 02:00:00              2.550              -2.350

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 22:00:00               -0.500                      0.0
2019-06-02 23:00:00                0.275                      0.0
2019-06-03 00:00:00                0.750                      0.0
2019-06-03 01:00:00                0.875                      0.0
2019-06-03 02:00:00                0.925                      0.0

                     is_estimated      y  location
ds
2019-06-02 22:00:00             0   0.00         A
2019-06-02 23:00:00             0   0.00         A
2019-06-03 00:00:00             0   0.00         A
2019-06-03 01:00:00             0   0.00         A
2019-06-03 02:00:00             0  19.36         A

[5 rows x 48 columns]
```

[4]:

```python
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
train_data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])
```

```
if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])


def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] / loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data = pd.DataFrame(), pd.DataFrame()
            for i in range(200):
                # get a part of the test set corresponding to i/100th part of the test set and shuffle
                num_bins = len(loc_test_set) // 200
                # set seed to i so that we get the same shuffle every time
                np.random.seed(i)
                current_bin = loc_test_set.iloc[i*num_bins:min((i+1)*num_bins, len(loc_test_set))].sample(frac=1)
                loc_tuning_data = pd.concat([loc_tuning_data, current_bin.iloc[:len(current_bin)//2]])
                loc_test_data = pd.concat([loc_test_data, current_bin.iloc[len(current_bin)//2:]])

            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
```

```
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        train_data = train_set
        test_data = test_set
        print("Shape of test", test_data.shape[0])

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Shapes of tuning and test 5000 5400 10400
```

[5]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
↪label="y", sample=None)
```

[6]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

## 2 Starting

[7]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
```

```python
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 92
Now creating submission number: 93
New filename: submission_93
```

[8]:
```python
predictors = [None, None, None]
```

[9]:
```python
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc],
        time_limit=time_limit,
```

```
        # presets=presets,
        # num_stack_levels=num_stack_levels,
        # num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        # num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_93_A"
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_93_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   236.42 GB / 315.93 GB (74.8%)
Train Data Rows:    29667
Train Data Columns: 38
Tuning Data Rows:    2000
Tuning Data Columns: 38
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 674.14552,
1195.53172)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
```

```
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132287.07 MB
        Train Data (Original)  Memory Usage: 11.21 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
```

```
Training model for location A…
```

```
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        36 features in original data used to generate 36 features in processed
data.
        Train Data (Processed) Memory Usage: 8.9 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Warning: use_bag_holdout=True, but bagged mode is not enabled. use_bag_holdout
```

```
will be ignored.
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif …
        -133.6185        = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.06s    = Validation runtime
Fitting model: KNeighborsDist …
        -133.2862        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBMXT …
        -108.6508        = Validation score   (-mean_absolute_error)
        1.14s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: LightGBM …
        -109.9947        = Validation score   (-mean_absolute_error)
        0.76s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: RandomForestMSE …
        -115.8985        = Validation score   (-mean_absolute_error)
        8.16s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: CatBoost …
        -119.3207        = Validation score   (-mean_absolute_error)
        3.71s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesMSE …
```

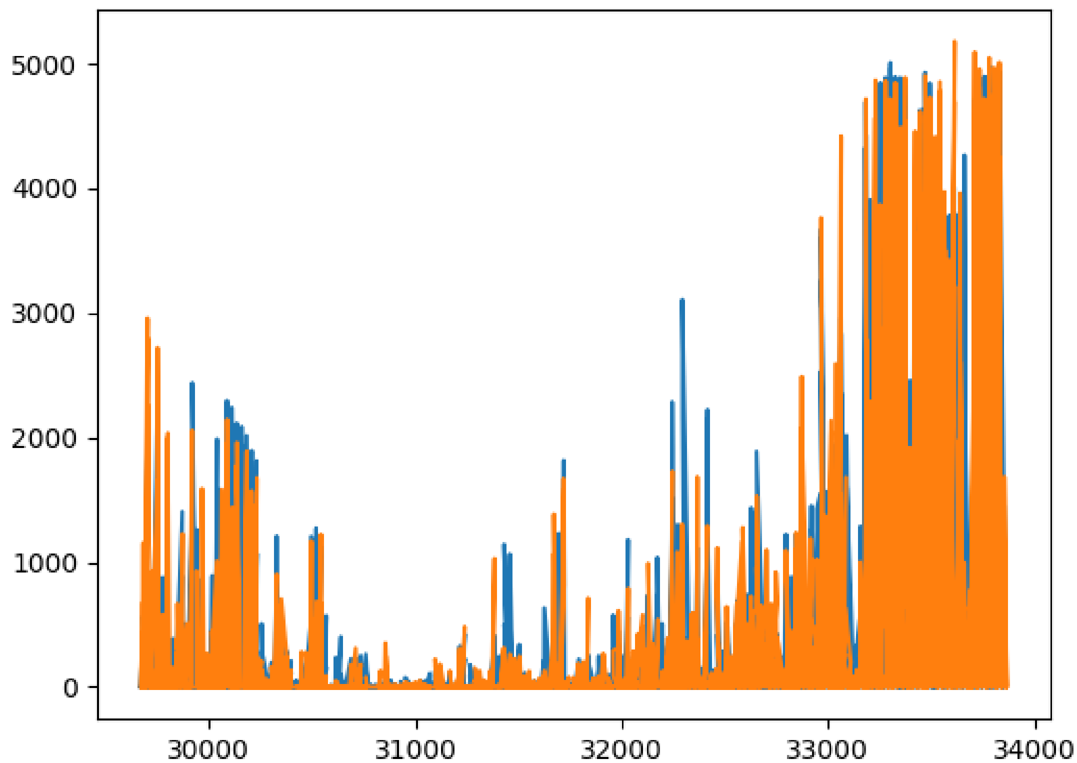
```
        -116.9235        = Validation score   (-mean_absolute_error)
        1.61s    = Training   runtime
        0.08s    = Validation runtime
Fitting model: NeuralNetFastAI …
        -127.3137        = Validation score   (-mean_absolute_error)
        28.4s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: XGBoost …
        -112.1297        = Validation score   (-mean_absolute_error)
        0.45s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch …
        -102.3995        = Validation score   (-mean_absolute_error)
        25.48s   = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBMLarge …
        -109.0342        = Validation score   (-mean_absolute_error)
        1.75s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -100.7955        = Validation score   (-mean_absolute_error)
        0.37s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 76.44s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_93_A/")
Evaluation: mean_absolute_error on test data: -108.96298462924184
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -108.96298462924184,
    "root_mean_squared_error": -300.9416589836497,
    "mean_squared_error": -90565.8821118313,
    "r2": 0.8842024557136907,
    "pearsonr": 0.940425816693644,
    "median_absolute_error": -0.6969462931156158
}

Evaluation on test data:
-108.96298462924184
```

[10]:

```
leaderboards = []
if use_test_data:
    lb = predictors[0].leaderboard(test_data[test_data["location"] == loc])
    lb["location"] = loc
    leaderboards.append(lb)
```

```python
    test_data[test_data["location"] == loc]["y"].plot()
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

```
                  model  score_test   score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0   WeightedEnsemble_L2 -108.962985 -100.795475        0.067740       0.044811
29.500449                0.003872                0.000688           0.366119
2        True         12
1         NeuralNetTorch -111.308892 -102.399462        0.030982       0.027988
25.484248                0.030982                0.027988          25.484248
1        True         10
2          LightGBMLarge -113.580632 -109.034159        0.013560       0.004905
1.745650                0.013560                0.004905           1.745650
1        True         11
3             LightGBMXT -116.092971 -108.650794        0.012526       0.006792
1.143641                0.012526                0.006792           1.143641
1        True          3
4               LightGBM -116.623948 -109.994701        0.006801       0.004439
0.760792                0.006801                0.004439           0.760792
1        True          4
5                XGBoost -117.405783 -112.129656        0.011609       0.006050
0.453634                0.011609                0.006050           0.453634
1        True          9
6          ExtraTreesMSE -118.563553 -116.923479        0.556726       0.084744
1.609335                0.556726                0.084744           1.609335
1        True          7
7        RandomForestMSE -119.504864 -115.898548        0.562806       0.099407
8.161116                0.562806                0.099407           8.161116
1        True          5
8               CatBoost -126.046294 -119.320659        0.035966       0.005841
3.711061                0.035966                0.005841           3.711061
1        True          6
9         KNeighborsDist -134.896380 -133.286181        0.035249       0.029379
0.033789                0.035249                0.029379           0.033789
1        True          2
10       NeuralNetFastAI -134.945192 -127.313714        0.115081       0.030549
28.403361                0.115081                0.030549          28.403361
1        True          8
11        KNeighborsUnif -135.049170 -133.618518        0.032794       0.058311
0.035364                0.032794                0.058311           0.035364
1        True          1
```




```
[ ]: loc = "B"
     predictors[1] = fit_predictor_for_location(loc)
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_93_B"

Training model for location B…

Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_93_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   236.42 GB / 315.93 GB (74.8%)
Train Data Rows:    29218
Train Data Columns: 38
Tuning Data Rows:    1600
Tuning Data Columns: 38
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
```

```
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, 0.0, 102.58516, 198.99359)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    131009.91 MB
        Train Data (Original)  Memory Usage: 10.91 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        36 features in original data used to generate 36 features in processed
data.
        Train Data (Processed) Memory Usage: 8.66 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
```

```
        To change this, specify the eval_metric parameter of Predictor()
Warning: use_bag_holdout=True, but bagged mode is not enabled. use_bag_holdout
will be ignored.
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif …
        -25.9296         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: KNeighborsDist …
        -26.2267         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBMXT …
        -19.2979         = Validation score   (-mean_absolute_error)
        0.63s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: LightGBM …
        -19.9849         = Validation score   (-mean_absolute_error)
        0.73s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: RandomForestMSE …
        -19.0379         = Validation score   (-mean_absolute_error)
        9.36s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: CatBoost …
        -20.4544         = Validation score   (-mean_absolute_error)
        0.73s    = Training   runtime
```

```
        0.0s     = Validation runtime
Fitting model: ExtraTreesMSE …
        -19.1835         = Validation score   (-mean_absolute_error)
        1.58s    = Training   runtime
        0.09s    = Validation runtime
Fitting model: NeuralNetFastAI …
        -20.7645         = Validation score   (-mean_absolute_error)
        27.97s   = Training   runtime
        0.03s    = Validation runtime
Fitting model: XGBoost …
        -19.3782         = Validation score   (-mean_absolute_error)
        0.55s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch …
        -14.8112         = Validation score   (-mean_absolute_error)
        17.03s   = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBMLarge …
```

[ ]:
```
if use_test_data:
    lb = predictors[1].leaderboard(test_data[test_data["location"] == loc])
    test_data[test_data["location"] == loc]["y"].plot()
    lb["location"] = loc
    leaderboards.append(lb)
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

[ ]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

[ ]:
```
if use_test_data:
    lb = predictors[2].leaderboard(test_data[test_data["location"] == loc])
    test_data[test_data["location"] == loc]["y"].plot()
    lb["location"] = loc
    leaderboards.append(lb)
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

[ ]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

## 3 Submit

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
```

```
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

[ ]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
↪"location"], left_on=["ds", "location"])

#test_data_merged
```

[ ]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
↪"prediction"] = past_pred
```

[ ]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")
```

```
    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
 ↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```

[ ]:
```
# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[ ]:
```
# Save the submission DataFrame to submissions folder, create new name based on␣
 ↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
 ↪index=False)
print("jall1a")
```

[ ]:
```
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

[ ]:
```
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
 ↪time_limit=60*10)
```

```
[ ]: # feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,␣
 ↪time_limit=60*10)
```

```
[ ]: display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
 ↪"autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
 ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
 ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
 ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
```

```
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```